Energy Introspector: Coordinated Architecture-Level Simulation of Processor Physics

William J. Song‡, Saibal Mukhopadhyay‡, Arun Rodrigues§ and Sudhakar Yalamanchili‡

‡School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
§Sandia National Laboratories, Albuquerque, NM 87185
‡wjhsong@gatech.edu, {saibal.mukhopadhyay, sudha.yalamanchili}@ece.gatech.edu, §afrodri@sandia.gov

***Abstract*—Increased power and heat dissipation in microprocessors impose limitations on performance scaling. Power and thermal management techniques coupled with workload dynamics cause increasing spatiotemporal variations in electrical and thermal stresses. The coupling between various physical phenomena (e.g., power, temperature, reliability, delay) will be critical to microarchitectural operations in future processors. Thus, we need modeling tools to enable the exploration of such physical interactions and drive development of microarchitectural solutions. This paper introduces a novel framework, *Energy Introspector* (EI), for the coordinated simulation of microarchitecture and physics models. The EI framework features flexible modeling of processor component hierarchy that enables simulating different microarchitecture and package designs. The proposed framework uses standardized interface to drive different implementations of physics models and captures their interactions. The EI supports parallel computation of models in anticipation of large-scale simulations (e.g., high core-count processors). We present a case study using the EI framework to assess reliability and performance tradeoffs with a full-system cycle-level simulation of an asymmetric chip multiprocessor (ACMP).**

I. Introduction

Sustaining microprocessor performance growth is challenged by physical limitations imposed by increased power and heat dissipation. Power and thermal limits are typically addressed using a variety of techniques including thread scheduling and migration, dynamic voltage-frequency scaling (DVFS), power gating (PG), etc. These techniques combined with inherent workload dynamics cause spatiotemporal variations of throughput, power, and temperature distributions. The resulting electrical and thermal stresses alter device properties and cause gradual and non-uniform device degradation across the chip. As industry moves to smaller feature sizes, performance will become increasingly dominated by the physics. The challenge is in understanding how the physics is manifested at the microarchitecture level. This requires simulation and modeling environment that can capture multiple distinct physical phenomena and the impact on the microarchitecture.

The principal modeling challenge is to accurately capture the interactions between multiple, distinct physical phenomena; *multi-physics modeling*. We refer to collective interaction of these phenomena and their architecture-level impact as *processor physics* and its modeling process as *coordinated architecture-level simulation*. In coordinated architecture-level simulation, computations of distinct physical phenomena occur in the same domain, interacting and affecting with each other. Thus, an integrated modeling environment must 1) include models of multiple distinct physical phenomena, 2) reconcile the concurrent computation of these models, and 3) accurately model the inter-dependence between the physical phenomena.

In this paper, we present a coordinated architecture-level simulation framework called *Energy Introspector* (EI). The goal of the EI framework is to facilitate microarchitectural exploration of tradeoffs between performance, energy, temperature, reliability, etc. In particular, we seek to design a universal, scalable, and flexible interface to incorporate new models as they are developed, and address practical issues of using the framework across various microarchitecture simulators. The EI framework features flexible modeling of processor component hierarchy that enables simulating different microarchitecture and package designs. Microarchitectural units are represented by modeling components (i.e., source components) and placed in a processor package (i.e., package component) via die partitions (i.e., partition components), leading to natural connection of microarchitecture, floorplanning, and package. The processor hierarchy is formed by modeling components including package, partition, and source-level components, and various physical models are attached to the modeling components at different levels of hierarchy. The EI provides a simulator application programming interface (API) to drive the computations of different physical models and automate their interactions via modeling components. Computed results are tagged with time information to correctly synchronize between multiple physical models and microarchitecture simulation. Thus, the EI framework enables us to construct the physical environment corresponding to microarchitecture to be simulated and reflect their inter-dependencies. The followings summarize the contributions of the proposed EI framework.

1) Compatibility: The EI framework does not rely on specific implementation of models. Any models in C/C++ can be integrated into the simulation framework. This is achieved by standardizing models via subclassing similar models into the same *library*.
2) Usability: The standardization of models facilitates the use of different models via the same virtual functions (interface). Thus, microarchitecture simulation use the same function calls to drive computations regardless of which models are actually used.

3) Flexibility: The framework can be configured to model any microarchitecture via the notion of *pseudo components*. The pseudo components form a tree hierarchy that is arbitrarily configurable, and library models are linked to appropriate pseudo components. This feature also enables the mixed use of multiple distinct models for different components to better represent the processor characteristics.
4) Coordination: The EI orchestrates the cross-reference of computed results between library models and linked pseudo components (e.g., between temperature and reliability). The data are tagged with time and period information for correct time synchronization among different simulators and models.
5) Scalability: The EI supports parallel execution for large-scale simulation via message passing interface (MPI) implementation. Microarchitecture simulators and coupled EI computations may run in parallel, and the EI itself can also run in parallel in multiple MPI ranks. In addition, each EI process in an MPI rank can be multi-threaded to perform independent computations in parallel.

The remainder of the paper is organized as follows. We first summarize related works in architecture-level modeling. Then, we propose the EI framework for coordinated architecture simulation and detail its implementation. Lastly, a case study as an exercise of reliability/performance tradeoff is presented based on transient degradation analysis of race-to-idle execution compared to normal execution. We argue that an infrastructure such as the EI is essential to explore such tradeoffs.

II. Model Categories and Related Works

In the past decade, the computer architecture community has put great effort into developing useful models and tools for the design space exploration of future microprocessor technologies and microarchitectures. The usefulness of models would be defined in terms of accuracy, speed, applicability to different technologies, and flexibility to explore various microarchitecture designs. Modeling tools both drive and limit research capabilities and directions at the same time. In this section, the architecture-level models of the most widely explored physical properties are reviewed, and the limitations of the models are presented. Our proposed framework accommodates all of these models. This section reviews many of the models available in the computer architecture community as background to their integration in the EI framework. Readers familiar with these models may skip ahead to Section III.

A. Architecture Modeling

The architecture-level design space exploration is largely based on the logical simulation of microarchitectural components and functions. With continued multi-core scaling and preference to architectural heterogeneity, building full-system environment requires more complex simulation capabilities and models; large core-count simulation with heterogeneous architectures, cache and coherence protocol modeling, on-chip network, and memory system. The individual component models can be implemented in numerous ways and vary in terms of functionality, detail, flexibility, speed, scalability, supported instruction set architecture (ISA), etc. In this paper, we focus on building scalable full-system environment for the co-simulation of physical characteristics rather than discussing the traits of individual component model. The challenge of architecture modeling is to configure a coherent combination of component models, depending on simulation purpose and features of individual models. The SST [1] and Manifold [2] infrastructures support configurable full-system architecture simulation environment by selectively building component models among a variety of integrated models including gem5 [3], Zesto [4], Qsim [5], Orion [9], DRAMsim [11], etc. In addition, the SST and Maifold both provide scalable parallel simulation based on the MPI that is a key to manycore simulations (Fig. 1), whereas a single-threaded use of heavy core models such as gem5 is practically limited to a few core simulation.

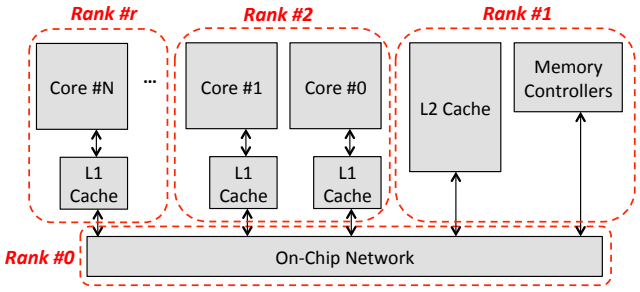

Fig. 1. Example of an MPI-based parallel manycore simulation environment based on Manifold discrete event simulation (DES) kernel.

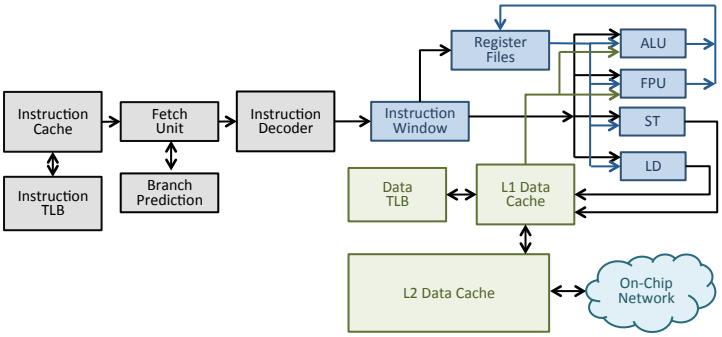

Fig. 2. Architectural decomposition into components within which architectural accesses and switching activities are characterized.

The architecture models generate statistics that differ by executed workloads and architectural configuration. The statistics are used to analyze the characteristics of input workloads and microarchitecture. The most important architectural statistics used for estimating physical phenomena are performance (or access) counts. They represent switching activities of architectural components as depicted in Fig. 2, which can be used to estimate dynamic energy dissipation in conjunction with circuit-level models. The counters at each component are differentiated by architectural behaviors such as data array reads/writes, tag array accesses, logical switching activities, etc. Typically high-level architecture models do not capture bit-level switching behavior due to modeling and simulation complexity unless specially focused on a component-level analysis.

B. Power Modeling

Energy or power is the most widely explored yet challenging physical phenomenon that must be associated with microarchitecture and workload features. There are largely two types of power models. First, an analytical model is based on experimental data measured from target processors. For instance, measured power is used to find weight factors of polynomial equations with respect to key metrics such as voltage, frequency, cycles per instruction (CPI), cache miss rate, etc [19]. The difficulty of using this approach lies on applying the model to new microarchitectures and technologies. The second approach relies on access counts acquired from architecture models. A circuit-level analysis of a modeled component is performed to estimate energy dissipation per access type (e.g., read, write). The dynamic energy is calculated as a sum of the product of access counts (C_{acc}) and per-access energy (E_{acc}) for each access type during the observation period (T_{observ}), and total power is sum of dynamic and leakage energies divided by time.

$$P(t) = \left\{ \sum_{T=1}^{N} E_{acc,T} \cdot C_{acc,T} + E_{leak} \right\} / T_{observ} \qquad (1)$$

This approach does not capture bit-level detail since it does not consider the number of switching bits thus energy variation at each access. However, because of flexibility to model various architecture configurations and applicability to different technologies, this approach is most popularly used for architecture-level power modeling as found in Cacti [6], McPAT [7], Wattch [8], Orion [9], DSENT [10], DRAMsim [11], etc. Cacti was one of the early architecture-level models that supports rapid estimation of energy, area, and timing of cache and memory structures [6]. Extending Cacti, McPAT added a variety of non-cache models such as instruction decoders, network, memory controller, functional units, etc [7]. Wattch was a widely used tool for dynamic power estimation based on a capacitance model and performance counters [8]. Orion is a detailed on-chip network model that estimates area and power based on a capacitance model of register-based FIFO buffers, router clocks, and physical links [9]. DSENT is an extended model of Orion and added an optical network model [10]. DRAMsim is a cycle-accurate memory simulator whose power calculation is based on Micron data sheets [11].

In full-system design space exploration, power is not only related to microarchitecture, workload, circuit design, and device technology but also advanced control techniques such as dynamic voltage and frequency scaling, power gating, thread migration, etc. Moreover, leakage power and operable clock frequency are functions of temperature, whose problems cannot be addressed without incorporating thermal modeling. Fig. 3 (a) an exponential dependency of leakage power on temperature and (b) an example of spatial variation of leakage power across 64-core processor die.

Despite strong coupling between power and temperature, few efforts are found to involve aforementioned modeling tools into research at the intersection of microarchitecture, workload, energy/power, and temperature due to complexity, speed, and efforts to integrate them. Thus, finding an effective way of incorporating useful physical models into design space exploration tools is a challenging task.

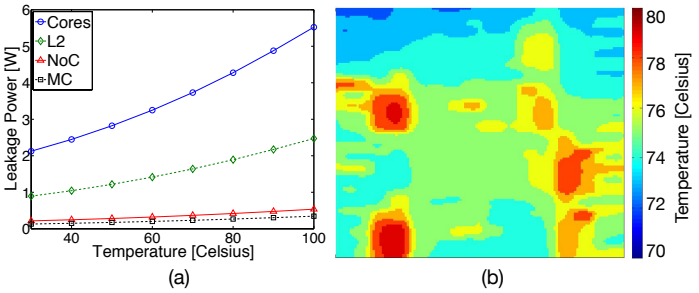

Fig. 3. (a) Temperature and leakage power dependency, and (b) spatial variation of temperature across 64-core processor die (Fig. 9, Table I), measured using McPAT [7] and HotSpot [13].

C. Thermal Modeling

Increased power density caused critical thermal problems in microprocessor packaging and cooling. Thermal modeling and management became another important topic for architecture research. HotSpot [13] is the most popular temperature model that brought thermal problems into architecture-level analysis and design space exploration. A processor package is designed as a volume of thermal grid cells that are comprised of thermal resistors and capacitors as depicted in Fig. 4. Thermal modeling is to solve differential equations derived from RC-composed thermal cells using the energy conservation property. Power dissipation is input to the source thermal cells. HotSpot divides the source layer into blocks called *floorplans* that is the unit block of power dissipation. Floorplan-level power is evenly divided into thermal cells that correspond to the location of floorplans. HotSpot supports temperature modeling of 2-dimensional packages comprised of substrate, source-layer silicon, thermal interface material, heat spreader, and heat sink layers [13]. Although HotSpot can configure vertically stacked source layers, it is not suitable to model 3-dimensional integrated circuits (3D ICs) due to the lack of important 3D package components such as through-silicon-vias (TSVs). 3D-ICE [14] supports inter-layer liquid cooling model of 3D ICs. It also utilizes floorplan-based power mapping similar to HotSpot. In both HotSpot and 3D-ICE as standalone tools, offline power traces are used to perform steady-state and transient power simulations. Such an interface does not capture power-temperature interactions and cannot be applied to design space exploration using dynamic control techniques such as DVFS.

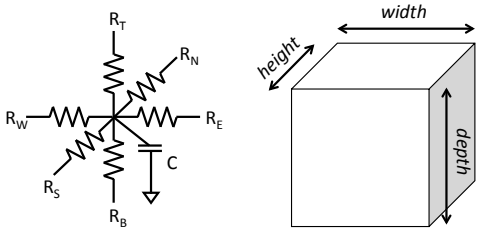

Fig. 4. A thermal cell model [13], [14].

D. Degradation and Reliability Modeling

Continued technology scaling and increased power density caused reliability concerns in microprocessors.

Moreover, increase of power and heat dissipation accelerates the degradation of silicon devices. Thus, early failed components threaten performability and reliability of processors. Traditionally reliability was characterized at the device-level. Srinivasan et al. in [16] presented architecture-level modeling of lifetime reliability. In this work, the lifetime reliability represented as mean-time-to-failure (MTTF) or failures-in-time (FITs) is modeled based on device-level characteristics. Each failure mechanism (e.g., electromigration, negative-bias temperature instability, etc.) is modeled as an exponential distribution with its own failure rate λ. Failure occurs when at least one failure mechanism takes into effect. The system-level reliability can thus be modeled as a series system of failure mechanisms; the system operates only when none of the in-series models fail. Since reliability is a function of various operation conditions such as switching activity, voltage, frequency, temperature, and stress time, reliability analysis cannot be performed without the co-simulation of multi-physics.

E. Coordinated Architecture Modeling

Interactions between microprocessor physical properties require a composite modeling environment. Earlier related works focused on characterizing a single target phenomenon, based on simplified assumption of other variables. However, such simplification inadvertently overlooks dynamic interaction between physical properties. For instance, power is a function of microarchitecture and workload. Power dissipation leads to temperature rise, and increased temperature again affects leakage power as shown in Fig. 3. Electrical and thermal stresses induced by switching activities cause degradation of silicon devices, which in turn affect timing and power. Processor control techniques place another degree of interaction by dynamically changing voltage and frequency levels, turning off cores, or applying different scheduling policies. The cyclic interactions between microarchitecture and physics cannot be captured by a single instance of a model. Thus, coordinated architecture modeling is an inevitable choice for accurate design space exploration.

A related work regarding coordinated architecture modeling is found in [19]. Bartolini et al. designed a virtual platform for the design space exploration of multicore processors by including the models of microarchitecture, power, temperature, and reliability. The authors also emphasized the importance of co-simulation of multiple physical properties with microarchitecture; a trace-driven cascaded simulation loses important information of cross-dependency, and thus simulation accuracy degrades [19]. Bartolini et al. composed a virtual platform on MATLAB/Simulink by incorporating Simics for architecture modeling, measurement-based power and thermal models tuned for Intel Xeon processors [19]. This work contributes coordinated modeling, but is target architecture specific and therefore difficult to extend.

III. Architecture-Level Modeling of Interactive Microprocessor Physics

Coordinated architecture simulation captures interactions among microprocessor physical phenomena and their impact on the microarchitecture. Such modeling enables the exploration of more complex problems. Separate use of models driven by offline traces cannot address the issues, where no physical interactions are reflected in pre-generated traces.

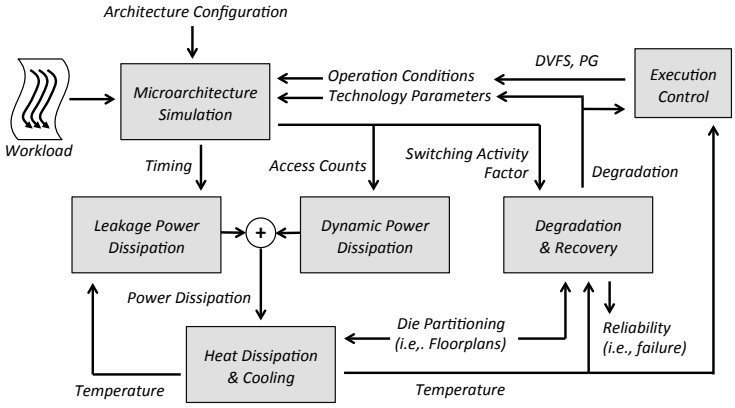

Fig. 5. Architecture-level modeling of interactive microprocessor physical phenomena and microarchitecture.

Figure 5 shows the model of architecture-level physical interactions. Given microarchitecture and technology, the execution of workload incurs switching activities at architecture components in data paths. Switching activities are represented as access counts and used to calculate dynamic power dissipation. Leakage power and temperature are assumed to be constant for short cycle periods. Power results are mapped onto thermal blocks (i.e., floorplans), and temperature is computed based on power dissipation during the cycle period. Stress conditions such as voltage, frequency, temperature, and time cause degradation/recovery effects [16], [18] and change device-level characteristics (i.e., technology parameters). Processor management involved in the cycle alters the execution of cores based on control decisions. In the next cycle, all computations of physical phenomena are affected by updated conditions and variables.

IV. Simulation Framework for Interactive Architecture and Physics Models

Designing a coordinated architecture simulation framework to model interactive physics and microarchitecture is a challenging and time-consuming process. There are various engineering problems that must be taken into account to implement a universal, flexible, and scalable simulation framework. We introduce a coordinated architecture simulation interface called *Energy Introspector* (EI) and describe its modeling methodology.

A. Design Motivation

We note that there already exist various modeling tools popularly used in the community. Great amount of effort has been invested to develop and validate the models, so utilizing those models is a reasonable start. We elaborate on developing sustainable framework that interacts with various models instead of discussing the accuracy or novelty of individual models. A variety of models are developed for different purposes, and no single model suffices all requirements (i.e., accuracy, speed, scalability, flexibility). The architecture community continues to develop new models or update

existing tools along with technology trends. Thus, a simulation framework must be compatible with various implementations of models and open to integration of new models. These features are distinct from the works of Bartolini et al. [19] where the presented virtual platform was based on a specific set of experimental models. In addition, the framework should not be constrained to certain microarchitecture or packaging design to limit the extent of design space exploration.

B. Standardization of Models

All models have different functionality and usage. Supporting all different features on the main interface eventually hinders the scalability of framework. In the EI framework, four categories called *model libraries* are defined; *energy*, *thermal*, *reliability*, and *sensor libraries*. The following summarizes the functions of the model libraries.

- Energy library:
  - Area, per-access energy estimation
  - Thermal design power (TDP), runtime power calculation
  - Runtime variable update
  - Models: McPAT, Orion, DRAMsim, IntSim, etc.
- Thermal library:
  - Static and transient temperature computation
  - Floorplan power mapping
  - 3D IC layer indexing
  - Runtime variable update
  - Models: 3D-ICE, HotSpot, etc.
- Reliability library:
  - Degradation estimation
  - Failure probability computation
  - Runtime variable update
  - Models: electromigration, NBTI, TDDB, HCI, etc.
- Sensor library:
  - Data read; read noise and delay
  - Models: Sensors of temperature and delay

Any model being integrated into the EI framework becomes a subclass of one of the libraries. Each library defines a set of virtual functions that an integrated model must provide. Models that fall into the same library are called with same virtual functions. The interface is thus standardized regardless of which model is used underneath. For instance, both HotSpot and 3D-ICE are wrapped into the thermal library, and temperature is calculated by calling the same virtual functions.

C. Processor Hierarchy and Pseudo Components

Different physical properties are characterized at different levels of design abstraction. For example, energy/power is characterized at architecture components with respect to performance counts. Temperature is computed at die floorplan and package levels. Reliability is typically computed at the floorplan level. We need a formation that enables us to couple these models in a coordinated and cohesive manner since they operate at distinct levels of modeling abstraction. We define the notion of *pseudo component hierarchy* in the EI framework.

The processor hierarchy is represented as pseudo component tree as in Fig. 6. A thermal library model is linked to *package*-level component that contains other components. A package-level component does not need to be the tree root, and there can be multiple roots to simulate different chip packages. The thermal library designates *partition*-level components that work similar to thermal floorplans of HotSpot [13] and 3D-ICE [14]. Reliability is characterized at this level. The pseudo component hierarchy is arbitrarily configurable, and there can be intermediate components between package and partition-level components, depending on processor hierarchy formation. The pseudo component tree continues under the partition-level component. Tree leaves are source-level components that energy library models estimate power dissipation with respect to access counts obtained from architecture simulation or measurement. The power results estimated at source-level components are summed upward the tree, and partition-level power is the sum of all child source components powers. This is a key difference between partitions and thermal floorplans, where floorplans themselves are power units in thermal models [13], [14]. Shrinking size of architectural components with continued technology scaling requires high resolution of thermal grids to have reasonable mapping between floorplans and thermal cells. In addition, it is practically impossible to place every architectural component on silicon die and find exact coordinates (e.g., width/length ratio, x-y location) in early-stage design space exploration. A better solution is to group architecturally and technologically similar components into the same partition and use it as a thermal floorplan. Similarly, results computed at the high-level components in the hierarchy such as temperature are applied to sub-tree components, if the data are utilized by the library models of child components.

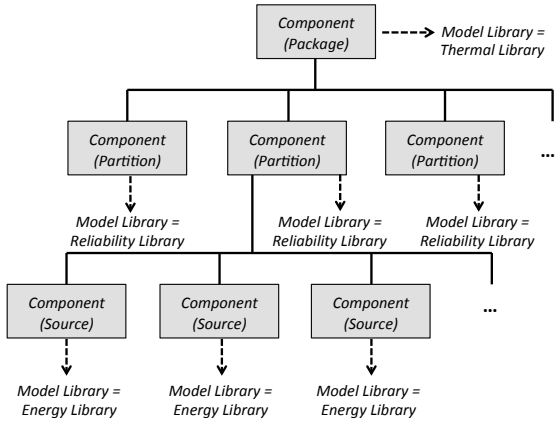

Fig. 6. An example of pseudo component hierarchy tree with components linked to model libraries.

Pseudo components are not only used to construct a processor configuration but also to match the discrepancy between architecture simulators and energy library models. In general, architecture simulators and models have different levels of details to define a configuration. A pseudo component represents a unit that the model is capable of estimating. It can be a logic gate, wire, architectural block (e.g., cache, functional unit), pipeline stage, or even entire processor package. Thus, pseudo components represent the modeling granularity. Binding model libraries to pseudo components enables mixed used of multiple libraries to better characterize the processor instead of relying on a sole model.

D. Coordination of Architecture Models and Simulation

A processor is modeled by the pseudo component hierarchy. The use of models is standardized via grouping models into libraries, and model libraries are instantiated at appropriate pseudo components. Computed results of library models are stored in the runtime data queue of pseudo components and synchronized through the pseudo component hierarchy. For instance, power calculated at source-level components is cumulatively added up the hierarchy. When data reference is required among pseudo components such as power for temperature computation, data queues provide necessary information requested by remote components and their models. The data queues are time synchronized such that all data are tagged with time and period information. Note that all transient data computed from models are sampled data (e.g., runtime power, transient temperature) during the observation period. For coordinated simulation, data produced by pseudo components must be time-synchronized. Parallel architecture simulation frameworks such as SST [1] and Manifold [2] are comprised of multiple architecture simulators (e.g., cores, caches, network, memory), and each processor component may run in a different logical process. Thus, computation at each process may occur out of order, and time synchronization check is critical for parallel simulation.

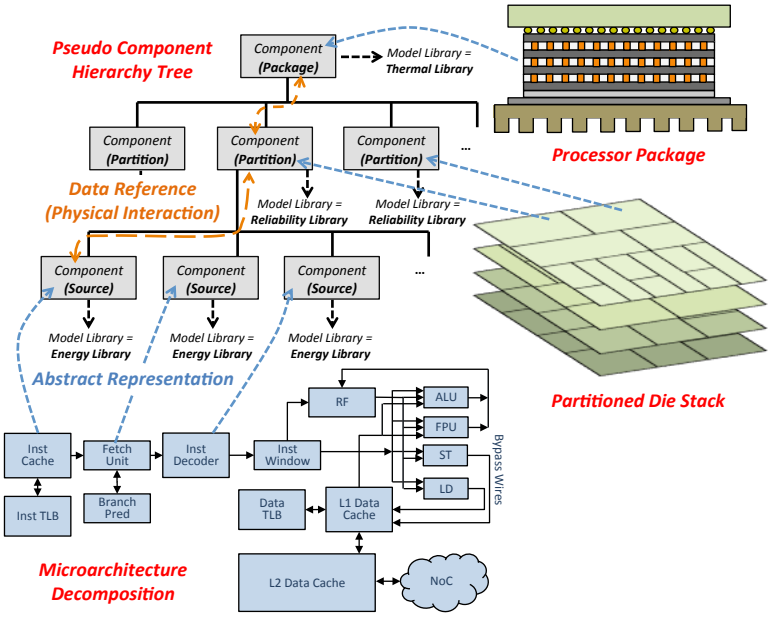

Fig. 7. Coordinated architecture simulation framework of Energy Introspector.

Fig. 7 shows the overview of the Energy Introspector framework that implements the flow graph in Fig. 5. Each source component is linked to one of the energy library models to estimate power dissipation with respect to access counts acquired from architecture simulators. The same virtual function call of the energy library is used at every source component to drive the library model and compute power. This function also updates parent components in the hierarchy tree. When temperature computation is called for a package component, it checks if all partition components have updated power data and they are synchronized with requested time and period tags. After temperature is calculated, partition components obtain updated temperature data. Since placement is not further defined for source components within a partition, it is assumed that source components have the same temperature value as the partition. When new data are pushed into data queues of components, necessary callback functions of library models are triggered to update the model states such as leakage power recalculation due to temperature change. Reliability is characterized at the partition level. It is represented as a failure probability using cumulative distribution function (CDF). Failure probability is a function of voltage, frequency, switching activity factor, stress time (e.g., on/off power gating period), and temperature. This process alters several technology parameters such as threshold voltage. The produced result is applied to other pseudo components in a similar way as temperature, and the callback functions of library models are used to update the variables and conditions. The altered variables and conditions affect the computations in the next simulation cycle.

E. Scalable Simulation for Manycore Processors

Similar to architecture simulations where a single-threaded simulation of heavy architecture models is practically limited to a few number of cores, the extent of complex physics simulation is also limited with a single-thread execution. The EI supports parallel execution of models over the MPI, as illustrated in Fig. 8. In multi-process simulation, the EI initiates a server listen thread. A simulator client can bind to only one server and is delegated to call the EI functions via MPI messages. Computation functions via clients are non-blocking such that architectural simulation can proceed while the requested computation is performed in the other rank. On the other hand, data request functions are blocking until they are returned from the server. Client simulators may manipulate computed results to apply scheduling or control policies.

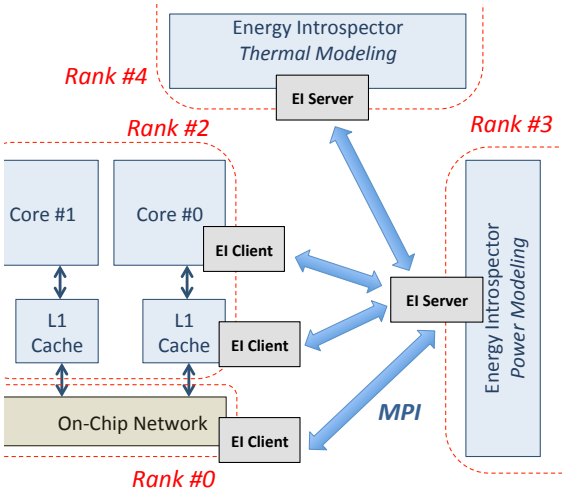

Fig. 8. Multi-process execution example of Energy Introspector over MPI interface.

F. Contributions

Compared with past related works, Energy Introspector has the following advantages:

1) The interface is compatible with various implementations of models. Thus, popular and already validated models can be utilized.
2) The standardized interface of models via grouping into model libraries enables simpler user interface. The interface does not depend on specific models.
3) Data is time-synchronized and can be referred by other components to support coordinated architecture simulation.
4) The component hierarchy can be configured in an arbitrary way to model different microarchitecture and package design.
5) Parallel execution is supported via the MPI implementation and multi-threading.

V. Application to Compound Analysis

A case study is performed with coordinated architecture simulation via the Energy Introspector. We use this simulation framework to characterize the impact of various failure mechanisms on processor reliability. Since degradation is a function of all stress variables such as switching activity, voltage, frequency, and temperature, *transient degradation* modeling requires a coordinated simulation framework. This study is distinct from previous related works that explored lifetime reliability problems based on abstract representation of microarchitecture and related physics [24], [25]. Those studies remained at high-level analyses such as increasing overall reliability via deploying duplicated structures or redundant cores. In this study, we present microscopic result than in [24], [25] by involving detailed microarchitecture, power, and temperature simulations. The case study is performed with race-to-idle (RTI) execution, and the resulting failure probability is compared against normal execution (NE).

TABLE I. Experiment Configuration for Coordinate Architecture Simulation

Configuartion	Description	
Simulator	Manifold 64-core simulation [2]	
Benchmarks	Multi-programmed execution of SPEC2006 suite	
Cores	Out-of-order	In-order
Core counts	16	48
Issue width	4	1
Reorder buffer size	128	N/A
L1 Cache	4-way assoc, 64-byte line, 32KB size	
L2 Cache	8-way assoc, 64-byte line, 256KB size, private L2	
Voltage/frequency levels	0.8V/2.0GHz for NE, 1.2V/4.0GHz for RTI	
Feature size	16nm technology projection to ITRS guideline	

A. Reliability Problem Description

Recent multicore processors utilize boosted execution of cores when hardware resources are underutilized and thus total power consumption is below the limit. The execution boost appears in various forms such as race-to-idle, Turbo Boost [20], Turbo Core [21], and computation sprinting [22] depending on boost level and stress releasing method.

In this case study, race-to-idle execution and its degradation consequence are assessed. The race-to-idle execution technique accelerates core execution by increasing clock speed and voltage level. Performance increase is achieved during the race period at the cost of higher power and heat dissipations that result in faster degradation. The boosted execution must be followed by idle period to mitigate thermal stress and degradation. The leakage power saving during the idle period is traded, and thus RTI execution is known to save energy over normal execution [20], [21], [22]. However, the reliability impact of RTI execution has not been addressed and is generally believed to be worse than normal execution.

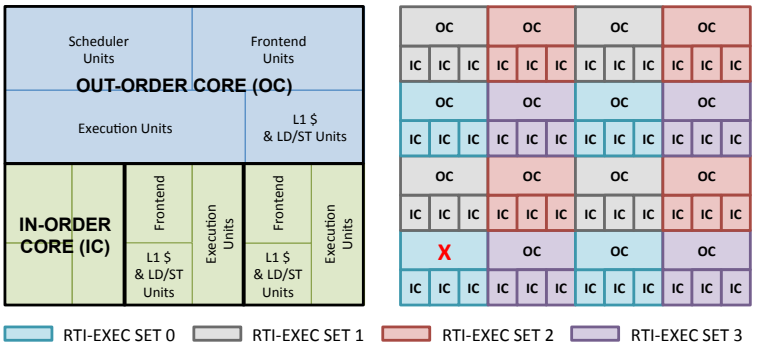

Fig. 9. Asymmetric processor die partition layout; cores are partitioned into pipeline stages of FE (frontend), SCH(schedule), EX(execute), and/or MEM(memory).

B. Failure Probability Modeling

Several failure mechanisms and degradation models are presented in [16], [17]. The processor is represented as a series system of failure mechanisms such that the processor operates only when none of the possible failure risks occur. In this case study, we characterize reliability as a *failure probability* using the CDF of exponential distribution functions. The following known failure mechanisms and their equations are used to calculate failure probability [16], [17].

- Electromigration (EM): Directional transport of electrons and metal atoms in interconnect wires leads to degradation and eventual failure [16].
- Time-dependent dielectric breakdown (TDDB): Wearout of gate oxide caused by continued application of electric field leads to electric short between gate oxide and substrate [16].
- Hot carrier injection (HCI): Electrons that capture sufficient kinetic energy overcome the barrier to gate oxide and cause threshold voltage shift and degradation [17].
- Negative bias temperature instability (NBTI): Holes trapped in the gate cause the threshold voltage shift and timing error. The switching between negative and positive gate voltages cause degradation and recovery of the BTI effects [16], [18].
- Stress migration (SM): Failure is caused by mechanical stress due to the difference between the expansion rates of metals.
- Thermal cycling (TC): Fatigue accumulates with temperature cycles.

The system-level failure occurs when at least one of aforementioned risks take effect. Therefore, the total failure

probability can be represented as a series system of failure mechanisms. Note that the failure probability is a non-decreasing function.

$$P_{total}(t) = 1 - P_0 \prod_{i=1}^{n} \prod_{r \in \text{Risks}} \begin{pmatrix} 1 - P_r(t_i - t_{i-1}) \\ |C_i(T_i, F_i, V_i, A_i, G_i) \end{pmatrix} \quad (2)$$

P_0 is the initial failure probability, and $P_r(t)$ is the failure probability of individual mechanisms, given the operational condition $C_i(T_i, F_i, V_i, A_i, G_i)$ as a function of temperature, frequency, voltage, switching activity factor, and power-gating state at time $t = t_i$. The failure probability at each computation interval $(t_i - t_{i-1})$ is multiplied to calculate the time-varying failure probability, based on the memoryless property of exponential distributions.

C. Reliability Implications of Race-to-Idle Executions

Table I shows the experiment configuration. A cycle-level x86 timing simulator, Zesto [4], is simulated with Manifold [2], and McPAT [7] and HotSpot [13] are used to estimate power and temperature with the Energy Introspector interface. Asymmetric 64-core layout is designed as in Fig. 9. Cores are grouped into 4 sets, and the RTI executions are applied to each set in a periodic round robin manner. The RTI execution is comprised of four phases as summarized in Table II and repeated throughout the simulation. Generally, Turbo Boost [20] or Turbo Core [21] increase clock speed not more than 50%, whereas computation sprinting [22] does in an order of magnitude over normal execution but with extra thermal material support. In this case study, we examine an intermediate case that doubles the speed of execution of cores for race mode (4.0GHZ, 1.2V) than normal execution (2.0GHz, 0.8V).

TABLE II. RACE-TO-IDLE EXECUTION PHASES

Phase	Race mode cores	Idle mode cores
Phase 0	Set 3, 0	Set 1, 2
Phase 1	Set 0, 1	Set 2, 3
Phase 2	Set 1, 2	Set 3, 0
Phase 3	Set 2, 3	Set 0, 1

Fig. 10 shows transient behavior of degradation calculated in failure probability. The transient behavior is observed at left-bottom out-of-order core, marked X in Fig. 9. Periodic race-to-idle executions with equal on/off duty cycle are made such that the race-to-idle and normal executions run equal amount of clock cycles. In this experiment, overhead for core power gating is not considered but is comparably smaller than race/idle periods [23]. Initially a core starts with accelerated execution. Accelerated execution during race period increases power and heat dissipation, leading to faster degradation. Degradation slope of RTI execution in Fig. 11 initially follows that of continuous race execution. After 10ms of execution, the core turns off. Given model equations presented in [16] and [17], failure probabilities due to some phenomena (i.e., EM, NBTI, TDDB, HCI) stay flat when cores are turned off, as shown in Fig. 11. Rather, recovery (i.e., NBTI) occurs during the idle period and decreases the degradation slope when the next race period starts, compared with that of continuous race mode. On the other hand, thermal stresses (i.e., SM, TC) remain as in Fig. 11 during the idle period, resulting in less but continued degradation overall as shown in Fig. 10. Turning off cores also helps reduce temperature increase across the processor die. Overall, the failure probability of race-to-idle execution shows similar result as normal execution.

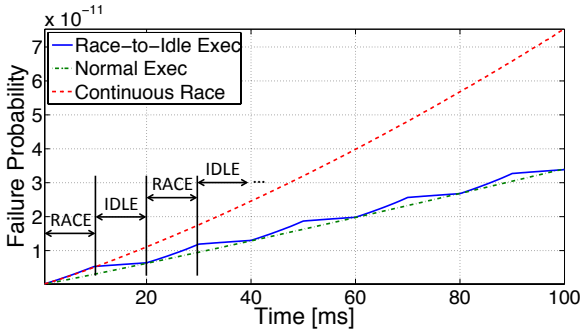

Fig. 10. Transient behavior of degradation calculated in failure probability for periodic race-to-idle and continuous executions.

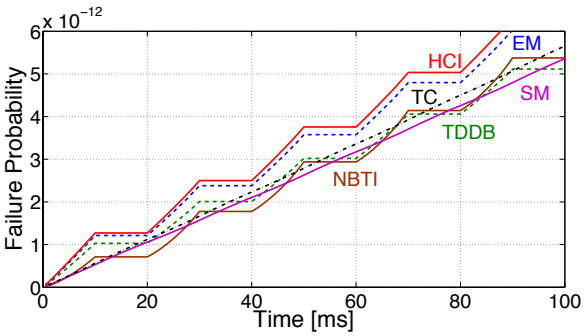

Fig. 11. Breakdown of transient degradation behavior of different failure mechanisms for race-to-idle execution.

The RTI execution performs differently depending on core types and executed benchmarks. Another experiment is performed to find a balance between race and idle periods while maintaining the failure probability equal to normal execution. First, continuous normal executions are applied to all cores, and transient failure probability of each core is sampled. This trace is used to control the RTI execution. Cores are run for fixed period and then turned off until failure probability becomes less than or equal to that of normal execution.

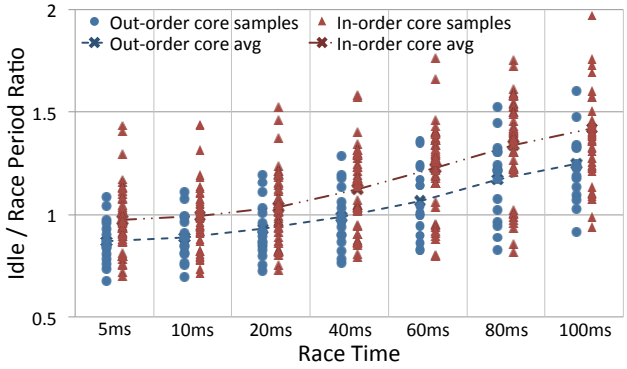

Fig. 12. Race and idle period balance with constrained failure probability equalized to the sampled trace of normal execution.

Fig. 12 shows the samples of idle to race period ratio. The ratio of 1.0 means the race and idle periods are equal while maintaining the same failure probability with normal execution. Since the race clock speed is twice faster than normal execution, 1.0 ratio means that the race-to-idle and normal executions operate the same number of clock cycles. Thus, the idle to race period ratio below 1.0 implies either less degradation or better performance. It is found that long race time has negative effect on degradation; longer idle period must follow to recover. Long race time creates hot spots on racing cores, leading to faster degradation and thermal coupling to neighboring cores. On the other hand, fast on/off switching puts high frequency terms in power dissipation that has less impact on temperature increase due to low pass filter characteristic of power to temperature relation [15]. Such power-temperature correlation affects degradation such that race-to-idle execution with faster on/off switching of cores perform better than normal execution. Also, difference is observed between two core types. The RTI applied to in-order cores is not as effective as out-of-order cores. Generally simple cores dissipate less power than complex cores for average behavior of benchmarks. With nature of low power and heat dissipations, in-order cores have slow increase of failure probability with normal execution. However, when the RTI is applied, accelerated execution causes faster degradation, and it takes more time to recover. Fig. 13 shows the inverse relation of time and failure probability for three different executions with same core type and workload. There is increasing time gap for equal difference of voltage and clock frequency. This means accelerating the execution of cores that dissipate low power and heat has worse impact on degradation than when they are normally executed, requiring lengthened recovery time that is translated as performance loss or more degradation. On the other hand, RTI applied to complex cores that dissipate more power and heat requires relatively shorter idle time than in-order cores. Similar analogy can be applied to workloads by differentiating their power usage. Thus, the RTI is better applied to complex cores than simple cores and power-consuming computing tasks than memory-bound workloads, not evening considering the complexity of scheduling long-latency memory instructions.

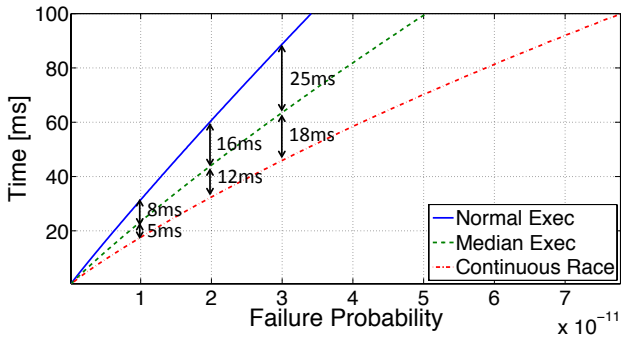

Fig. 13. Inverse plot of time vs failure probability; normal execution (2.0GHz, 0.8V), median execution (3.0GHz, 1.0V), and continuous race (4.0GHz, 1.2V). Relative recovery time increases from median to normal execution than from continuous race to median execution.

VI. Conclusion

Addressing physical challenges has become an important part of architecture analysis and design space exploration. Due to modeling complexity, interactions among physics phenomena have been neglected in the past architecture-level analyses. The coordinated architecture simulation framework enables correct microprocessor modeling and compound analyses.

Acknowledgment

The authors gratefully acknowledge the grants from Semiconductor Research Corporation (Task No. 2084.001), SRC/IBM Graduate Fellowship Program, and Sandia National Laboratories.

References

[1] G. Henry et al., "SST: A Simulator for Exascale Co-design," *Exascale Research Conf.*, Apr. 2012.

[2] J. Wang et al., "Designing Configurable, Modifiable and Reusable Components for Simulation of Multicore Systems," *PMBS*, Nov. 2012.

[3] N. Binkert et al., "The gem5 Simulator," *Comp. Arch. News*, May 2011.

[4] G. Loh et al., "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration," *ISPASS*, Apr. 2009.

[5] C. Kersey et al., "A Universal Parallel Frontend for Execution Driven Microarchitecture Simulation," *RAPIDO*, Jan. 2012.

[6] S. Thoziyoor et al., "A Comprehensive Memory Modeling Tool and Its Application to The Design and Analysis of Future Memory Hierarchies," *ISCA*, June 2008.

[7] S. Li et al., "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," *MICRO*, Dec. 2009.

[8] D. Brooks et al., "Wattch: A Framework for Architecture-Level Power Analysis and Optimization," *ISCA*, June 2000.

[9] A. Kahng et al., "Orion 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," *DATE*, Apr. 2009.

[10] C. Sun et al., "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," *NoCs*, May 2012.

[11] P. Rosenfeld et al., "DRAMsim2: A Cycle Accurate Memory System Simulator," *Comp. Arch. Letters*, Jan.-June 2011.

[12] Sekar et al., "IntSim: A CAD Tool for Optimization of Multi-Level Interconnect Network," *ICCAD*, Nov. 2007.

[13] W. Huang et al., "HotSpot: A Compat Thermal Modeling Methodology for Early-Stage VLSI Design," *TVLSI*, Nov. 2006.

[14] A. Sridhar et al., "3D-ICE: Fast Compact Transient Thermal Modeling for 3D ICs with Inter-Tier Liquid Cooling," *ICCAD*, Nov. 2010.

[15] M. Cho et al., "Thermal System Identification: A Methodology for Post-Silicon Characterization and Prediction of The Transient Thermal Field in Multicore Chips," *SemiTherm*, Mar. 2012.

[16] J. Srinivasan et al., "Lifetime Reliability: Toward An Architectural Solution," *Micro*, May-June 2005.

[17] M. White et al., "Microelectronics Reliability: Physics-of-Failure Based Modeling and Lifetime Evaluation," *NASA JPL Publication*, 2008.

[18] W. Wang et al., "The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis," *TVLSI*, Feb. 2010.

[19] A. Bartolini et al., "A Virtual Platform Environment for Exploring Power, Thermal, and Reliability Management Control Strategies in High-Performance Multicores," *GLSVLSI*, May 2010.

[20] E. Rotem el al., "Power-Management Architecture of The Intel Microarchitecture Code-Named Sandy Bridge," *Micro*, Mar.-Apr. 2012.

[21] A. Branover et al., "AMD Fusion APU: Llano," *Micro*, Mar.-Apr. 2012.

[22] A. Raghavan et al., "Computational Sprinting," *HPCA*, Feb. 2012.

[23] T. Xu et al., "Decoupling for Power Gating: Sources of Power Noise and Design Strategies," *DAC*, June 2011.

[24] J. Srinivasan et al., "Exploiting Structural Duplication for Lifetime Reliability Enhancement," *ISCA*, June 2005.

[25] L. Huang et al., "Characterizing The Lifetime Reliability of Manycore Processors with Core-level Redundancy," *ICCAD*, Dec. 2010.